VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

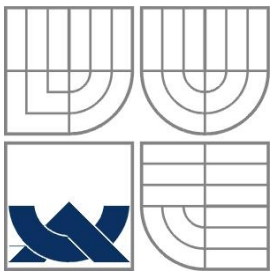
KNIHOVNA PRO OVLÁDÁNÍ IP KAMER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

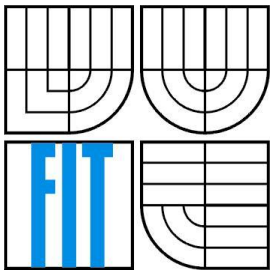
AUTOR PRÁCE
AUTHOR

JAN SCHMIED

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KNIHOVNA PRO OVLÁDÁNÍ IP KAMER

IP CAMERAS CONTROL LIBRARY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN SCHMIED
AUTHOR

VEDOUCÍ PRÁCE — ING. RADIM LUŽA
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se zabývá návrhem a implementací knihovny, která umožní ovládání IP kamer a přijímat multimediální streamy. V práci jsou nejprve stručně představena standardní rozhraní IP kamer a detailně popsáno rozhraní VAPIX. Na tomto rozhraní jsou také demonstrovány ukázky. V praktické části se zabývá návrhem a implementací knihovny s využitím knihoven z FFmpeg a cURL. V závěru je provedeno měření, které zobrazuje výkonnost knihovny.

Abstract

This thesis describes the design and implementation of a library that allows control of IP cameras and receives multimedia streams. The thesis briefly introduces standard interfaces of IP cameras and describes in detail VAPIX interface. This interface is also demonstrated by examples. The practical part deals with the design and implementation of the library using the FFmpeg and cURL libraries. Library is subject of a measurement that shows its performance.

Klíčová slova

IP kamera, rozhraní IP kamer, VAPIX, multimediální streamy, FFmpeg, cURL, RTSP, HTTP

Keywords

IP camera, IP cameras interfaces, VAPIX, multimedia streams, FFmpeg, cURL, RTSP, HTTP

Citace

Jan Schmied: Knihovna pro ovládání IP kamer, bakalářská práce, Brno, FIT VUT v Brně, 2012

Knihovna pro ovládání IP kamer

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radima Lužy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jan Schmied
14. května 2012

Poděkování

Tímto bych rád poděkoval svému vedoucímu Ing. Radimu Lužovi za vedení a pomoc při psaní této práce a také za zpřístupnění potřebného vybavení.

Obsah

1 Úvod

IP kamery v dnešní době pomalu vytlačují běžné analogové CCTV[1] kamery. Výhodou IP kamer je, že nepotřebují žádné přídavné karty do počítače ani žádné enkodéry. Připojují se totiž přímo ke stávající počítačové síti. Tím se ušetří náklady za natažení nové kabeláže pro analogové kamery. Vyměňují se ale i stávající zavedené analogové systémy, protože digitální IP kamery mají vyšší kvalitu obrazu. Jelikož všechny tyto kamery slouží převážně pro monitorování objektů a zajišťování bezpečnosti, obsahují IP kamery rozšíření, které by nebylo možné použít na analogových kamerách např. detekce pohybu. Právě proto získávají IP kamery v poslední době na popularitě.

Ke kamerám většinou výrobce dodává svoji aplikaci, která umožňuje ovládání kamer, které pochází z jeho produkce. Tím vzniká problém. Pokud bychom koupili kameru od jiného výrobce, zřejmě ji nebude možné začlenit do současného systému.

Tato bakalářská práce si klade za cíl vytvořit knihovnu, která umožní snadné vytváření aplikací, které budou IP kamery ovládat. Umožní také zapojovat do systému kamery od různých výrobců. To je možné pomocí standardizovaných rozhraní kamer, která ale nejsou příliš známá. Tato rozhraní sice nebudou obsažena v knihovně, ale budou stručně představena a knihovna se přizpůsobí pro jejich doplnění. Umožní také přijímat video z kamery a jeho dekompresi pro případnou následnou analýzu.

Nejprve se zaměřím na představení standardů pro kamery a popis možností rozhraní VAPIX2 a VAPIX3 od společnosti AXIS. Dále pak na návrh knihovny, která se snaží být co nejvíce rozšiřitelná. Ve čtvrté kapitole se více zaměřím na knihovny, které jsem použil k implementaci a stručně popíši jejich přínos a jak je používat. Přestože jsou knihovny rozšířené, jejich dokumentace je velmi strohá a neobsahuje všechny potřebné informace.

Pátá kapitola je zaměřena na implementaci knihovny a popisuje problémy, které při implementaci vznikly a jak byly řešeny. V šesté kapitole se potom seznámíme s překladem knihovny, používáním knihovny a doporučenými postupy. Zde také představím výsledky měření, které zachycují výkonnost knihovny a kamery.

[1] Closed-circuit television – uzavřený televizní okruh – kamerový systém

2 Rozhraní IP kamer

V této části jsou popsána některá rozhraní pro práci s IP kamerami a jejich vlastnosti a možnosti. Porovnám jednotlivé standardy pro práci s IP kamerou, jejich klady a zápory. Práce je zaměřena převážně na rozhraní VAPIX, jehož implementace je součástí bakalářské práce. Budou prezentovány schopnosti rozhraní VAPIX a ukázky použití tohoto rozhraní.

2.1 Dostupná rozhraní

Rozhraní pro ovládání IP kamer je mnoho a dá se říci, že každý výrobce si vymyslel vlastní rozhraní vhodné pro jeho produkty a podle jeho stylu. Někdy dokonce nemá kamera žádné rozhraní a je nutné používat ovládání z webového rozhraní. Každý výrobce tak má odlišný způsob práce s kamerou a je nutné pro každého výrobce udělat zvláštní rozhraní aplikace pro práci s jeho kamerami. Toto je pro vývojáře velmi nepříjemné. Proto vzniklo v posledních letech několik průmyslových standardů. Všechny níže uvedené standardy používají pro komunikaci s kamerou protokol HTTP nebo HTTPS. Standardy většinou neřeší přenos videa. Ten je zajištěn protokolem RTP a RTSP jako inicializátorem přenosu.

2.1.1 PSIA (Physical Security Interoperability Alliance)

Byl vytvořen v roce 2008 společnostmi IBM, Cisco a dalšími. Komunikace s kamerou probíhá pomocí URL. Na kameře běží HTTP server, který přijímá požadavky. Ty v path obsahují příkaz, který se má vykonat a v parametrech potřebná data. Výhoda tohoto standardu je malý objem dat, který je nutno přenést pro provedení příkazu.

2.1.2 ONVIF (Open Network Video Interface Forum)

Standard z roku 2008 od společností AXIS, BOSCH a Sony. S kamerou se komunikuje pomocí SOAP zpráv. Velkou nevýhodou je, že se každý požadavek musí opatřit SOAP hlavičkou a obálkou. V porovnání s PSIA , kde může mít požadavek desítky bajtů, se u ONVIF dostáváme k hodnotám v řádech kilobajtů. To je velmi nevhodné při malých rychlostech sítě, po které navíc přenášíme i video z kamery.

2.1.3 VAPIX

Rozhraní pro kamery AXIS. Aktuálně jsou podporovány 2 verze. Verze 2 z roku 2003 (aktuální 2.14 z roku 2007) pro starší zařízení s firmware verze 4.x. Verze 3 z roku 2008 obsahuje pouze drobná vylepšení a zjednodušuje formát cesty v URL, není kompatibilní s verzí 2 a je podporována zařízeními s firmware 5.x. Tyto zařízení také podporují ONVIF. Komunikace je podobná jako v případě PSIA, URL obsahuje v path příkaz a v parametrech data.

2.2 Vlastnosti rozhraní

Zde popisované vlastnosti a ukázky jsou z rozhraní VAPIX. PSIA a ONVIF nejsou v této práci dále řešeny a předpokládá se, že tyto vlastnosti podporují také, byť možná jinak a za použití odlišných postupů. V příkladech je vždy uveden příkaz pro VAPIX V2 a V3.

2.2.1 Inicializace

Kameru samotnou není třeba jakkoliv inicializovat, je funkční od okamžiku, kdy je zapnuta. Komunikace probíhá přes bezstavové HTTP parametry se tedy vždy přenesou znovu. Inicializace je ovšem potřebná na straně aplikace, aby aplikace věděla, co kamera podporuje, jak je nastavena a co má aplikace za přístupová práva. V případě V2 lze přístupová práva zjistit pouze tak, že se pokusíme číst parametry z různých úrovní přístupu, dokud kamera neodpoví, že pro tyto parametry nemáme oprávnění. V3 podporuje získání seznamu parametrů ve formátu XML, kde je u každého parametru uvedena potřebná úroveň přístupu a aktuální hodnota.

Parametry obsahují užitečné informace o aktuálním nastavení kamery, např. jakou má IP, nastavení objektivu, polohu natočení kamery, registrované události, rozlišení videa atd. Tyto informace jsou užitečné pro některé aplikace, pro jiné však můžou být zbytečné. Čím větší oprávnění má přihlášený uživatel, tím více parametrů je vypsáno.

Pro autentizaci uživatelů kamery se používá HTTP Authentication v režimu Basic. Parametry také obsahují informaci o tom, jakou verzi VAPIX podporuje kamera, tato informace je ale poněkud zbytečná, protože každá verze používá jiný formát URL a tak je nutné vědět, jakou verzi kamera podporuje ještě před čtením parametrů.

Příklad:
V2: *http://<servername>/axis-cgi/<access-lvl>/param.cgi?action=list&[<param>=<val>...]*
V3: *http://<servername>/axis-cgi/param.cgi?action=list[&<param>=<val>...]*

2.2.2 Nastavení parametrů

Kameře lze také upravovat různé parametry. Úprava může proběhnout kdykoliv, je tedy jedno, jestli při inicializaci, nebo později. Nastavování parametrů podléhá oprávnění, je tedy nutné být přihlášen jako uživatel s požadovaným oprávněním pro nastavení určitého parametru. Například úroveň viewer může číst rozlišení kamery, ale pro nastavení je třeba oprávnění operátora. Zde opět vyvstává problém v aplikaci, jak určit správnou úroveň přístupu a zobrazit požadované nabídky a hodnoty k editaci, pokud nebyl vyřešen již při inicializaci metodou pokus-omyl.

Hodnoty se nastavují velmi podobně, jako se čtou. Jen se změní akce na změnu případně přidání, smazání. Přidávání a smazání hodnot je možné pouze v případě, že se jedná o dynamické parametry, jako je například akce pro událost. Ostatní parametry můžou být pouze upraveny. Pro úpravu některých parametrů je ovšem nutné použít metody k tomu určené a nelze je nastavit jako většinu parametrů. Mezi tyto parametry patří například nastavení data a času kamery.

Příklad:
V2: *http://<servername>/axis-cgi/<access-lvl>/param.cgi?action=update&[<param>=<val>...]*
V3: *http://<servername>/axis-cgi/param.cgi?action=update[&<param>=<val>...]*

2.2.3 Správa

Rozhraní obsahuje také mnoho funkcí pro administraci kamery. Je možné přidávat, upravovat a odebírat uživatele v kameře a řadit je do různých skupin. Dále je možné resetovat kameru, nebo provést factory default, při kterém se všechny parametry nastaví na své výchozí hodnoty kromě IP adresy zařízení. Hard factory reset navíc nastaví IP na výchozí hodnotu. Kameru je pak třeba znovu v aplikaci inicializovat.

Další možnost je aktualizovat firmware a uložit zálohu nastavení a obnovení uloženého nastavení. Pro aktualizaci firmware a obnovu se použije HTTP POST v kódování multipart/form-data který obsahuje data souboru s firmwarem nebo zálohou.

Užitečné pro správu je také výpis systémových logů, přístupových logů a stavu serveru. Pro všechny operace, které mění funkčnost kamery, je potřeba mít oprávnění správce, jinak operace skončí chybou.

Příklad restartu kamery: *V2: http://<servername>/axis-cgi/admin/restart.cgi* V3: *http://<servername>/axis-cgi/restart.cgi*

2.2.4 Získání snímků

Mezi základní vlastností kamery patří nepochybně i získání snímku například v případě, kdy je třeba vyfotit zloděje, který se zrovna kouká do kamery. Je možno zachytit snímek ve formátech JPEG a BMP. Pro zachytávání snímků jsou k dispozici metody, které ve své odpovědi obsahují data obrázku.

Formát BMP se posílá z kamery bez komprese, není tedy podporována bezztrátová komprese BMP. Protože je takto pořízený snímek velký, jeho odeslání po pomalé síti zabere mnoho času, ale na druhou stranu je ve vysoké kvalitě, která je vhodná pro následnou analýzu obrazu.

Pro formát JPEG jsou k dispozici 2 metody. Jedna z nich nevyžaduje žádné parametry a vrací snímek podle parametrů nastavených v kameře (tyto parametry lze upravit). Druhá metoda přijímá všechny parametry vlastností snímku. Tak je možné specifikovat kompresi, rozlišení obrazu, text ve snímku, otočení snímku apod. pokud nejsou parametry specifikovány, jsou použity výchozí hodnoty jako v případě první metody.

Příklad získání JPEG snímku 640x480 a kompresí 80: *V2 i V3: http://<servername>/axis-cgi/jpg/image.cgi?resolution=640x480&compression=80*

2.2.5 Video

Získávání videa je základní vlastností kamery. AXIS kamery umožňují získat video ve dvou formátech M-JPEG a MPEG4 (nejnovější kamery i H264). Pro video je také možné specifikovat parametry pro jeho kvalitu a velikost, samozřejmě platí, že čím větší kvalita, tím větší jsou jednotlivé snímky a tím déle trvá jejich přenos. Toto je důležité především pro M-JPEG, kde může pak docházet ke zpoždění snímků.

MPEG4, který se přenáší pomocí protokolu RTP a kvalita je kontrolována protokolem RTCP, není nutné dbát tolik na parametry videa jako v případě M-JPEG. V případě přenosu pomocí skupiny protokolů RTSP, RTP a RTCP je pravidelně klient dotazován na kvalitu obdrženého videa a parametry jsou upraveny tak, aby byl přenos efektivnější.

2.2.5.1 M-JPEG

Formát M-JPEG je velmi jednoduchý. Odpověď od serveru obsahuje proud snímků ve formátu JPEG. Server je posílá v HTTP kódování *multipart/x-mixed-replace; boundary=<boundary>* kde --<boundary> odděluje jednotlivé snímky a každá část navíc obsahuje ve své hlavičce informaci o typu dat, v tomto případě *image/jpeg*, a délku části, což je zde velikost snímku v bajtech. Proud je možné získat opět dvěma způsoby jako v případě JPEG snímků. Jedna metoda nepřijímá žádné parametry a používá výchozí hodnoty, které lze změnit v parametrech kamery. Druhá metoda přijímá většinu

parametrů jako v případě JPEG snímků ale navíc obsahuje parametry pro video, jako například počet snímků za sekundu nebo celkový počet snímku, které chceme obdržet.

Formát M-JPEG je velmi jednoduchý ale má velkou nevýhodu v porovnání s formátem MPEG4. Je totiž nutné vždy přenést celý snímek. To způsobí velké zatížení sítě a nízkou kvalitu videa, nebo nižší počet snímků za sekundu. Také neobsahuje žádnou kontrolu kvality a rychlosti přenosu a tak může dojít ke zpoždění snímků.

Příklad zahájení M-JPEG streamu 5 snímku za sekundu: *V2 i V3: http://<servername>/axis-cgi/mjpg/video.cgi?fps=5*

2.2.5.2 MPEG4

Formát MPEG4 lze z kamery získat pomocí RTSP. Nejprve je nutné získat informace o proudu videa pomocí SDP (Session Description Protocol). Ten obsahuje informace pro příjem streamu jako například formát videa, adresa videa na RTSP serveru, délka videa apod. SDP data je možné získat buď z rozhraní kamery, nebo přímo z RTSP serveru při znalosti názvu videa přes metodu RTSP DESCRIBE.

Pokud použijeme SDP, můžeme poslat nastavení pomocí RTSP SETUP jak chceme stream přijímat. Kamera podporuje unicast, multicast a unicast přes TCP. Jakmile je stream nastaven, stačí zavolat RTSP PLAY a kamera začne posílat data na port, který jsme nastavili pomocí SETUP. Stream je také možné pozastavit RTSP PAUSE a opět spustit RSTP PLAY, nastavení zůstane stejné, pouze přestane kamera posílat data.

Pokud chceme ukončit příjem, musíme použít metodu RTSP TEARDOWN. Kvůli tomu, že je stream většinou posílán přes UDP, kamera by nevěděla, že již data nepřijímáme. To by mohla zjistit nejdříve při neobdržení několikáté zprávy o stavu přenosu RTCP. Touto metodou oznámíme, že již nemáme o video zájem a kamera přestane posílat video a uvolní zdroje, které stream blokoval. Po ukončení streamu jsou všechna nastavení tohoto streamu na kameře smazána, a pokud jej chceme opět přijímat, je nutné stream opět pomocí RTSP SETUP nastavit.

Velká výhoda streamování pomocí RTSP je, že data jsou posílána pomocí UDP, které je rychlejší než bezpečnější TCP. Kvalita linky je kontrolována zprávami RTCP (RTP Control Protocol), ve kterých je klient dotazován na kvalitu přenosu. Server na základě odpovědi klienta upraví hodnoty.

2.2.6 Audio

Některé kamery obsahují také mikrofon, anebo konektor pro připojení externího mikrofonu pro monitorování zvuku v místnosti. Dostupnost audia lze zjistit z parametrů kamery, viz Inicializace. Tam se také dozvíme, jaké audio kodeky jsou podporovány. Mezi ně bohužel nepatří populární MP3, ale jedná se spíše o telekomunikační kodeky G.711 a G.726.

Při použití kodeku G.711 se při kompresi používá algoritmus μ-law a datový tok je 64kbit/s. V případě kodeku G.726 si můžeme vybrat ze dvou datových toků, buď 32kbit/s, nebo 24kbit/s. V případě toku 24kbit/s jde prakticky o kodek G.723, a pro 32kbit/s jde o kodek G.721. Oba kodeky, G.723 a G.721, jsou součástí kodeku G.726.

Pro příjem si opět můžeme zvolit způsob, jak chceme audio data přijímat. Možnosti jsou podobné jako v případě M-JPEG. První možností je nechat si od kamery poslat zvuk jako jeden nekonečný proud bloků, kdy má každý 240bajtů. Druhá možnost je jako pro M-JPEG si nechat posílat bloky odděleně. Každý opět obsahuje 240bajtů, ale jsou kódovány v HTTP *multipart/x-mixed-replace; boundary=<boundary>* kde --<boundary> odděluje jednotlivé bloky.

Příklad zahájení příjmu zvuku jako jeden proud:
V2 i V3: http://<servername>/axis-cgi/audio/receive.cgi?httptype=singlepart

2.2.7 PTZ (Pan-Tilt-Zoom)

Kamery se systémem PTZ obsahují také rozhraní pro otáčení, naklánění a zoomování. Do PTZ je také zahrnuto ovládání ostření, clony a jasu. Rozhraní je velmi komplexní a obsahuje spoustu možností jak s kamerou pohybovat a také umožňuje uložit aktuální polohu pod určitým názvem a na tuto polohu se lze vrátit. Poloha ovšem nemusí být naprosto přesně dodržena, obzvláště v případě velkého zoomu, se může trochu odlišovat. Nepřesnost je dána vůlí v mechanizmu PTZ (převody, servomotory).

Před používáním PTZ je možné nainstalovat PTZ ovladač. To platí pro kamery, které nejsou vybaveny systémem PTZ ale jsou uzpůsobeny pro jeho dodatečnou instalaci. Taková kamera obsahuje sériový port, do kterého se PTZ zařízení připojí. Ovladač se do kamery nahraje stejným způsobem jako firmware nebo nastavení. Potom se nastaví, který ovladač má kamera použít, a číslo sériového portu.

Pohybovat kamerou lze v několika režimech. První režim je relativní a kamera se pohybuje relativně vůči své aktuální poloze (resp. aktuálním hodnotám), například otočení kamery o 2° vlevo a 10° dolů. Další režim je absolutní, při tomto režimu se kamera pohybuje relativně vůči výchozí poloze, tedy poloze 0, například otočení kamery na úhel 20° napravo a 70° dolů. Další režim je kontinuální, ten je podobný relativnímu režimu, ale místo úhlů se udává rychlost pohybu, například pohybuj kamerou doprava rychlostí 10 a nahoru rychlostí 5 (jednotka rychlosti není specifikována, zřejmě však stupně za sekundu).

Další režimy jsou simulované a jsou určeny pro pohodlné ovládání uživatele. Hlavní je režim automatického ostření a clony. Pokud kamera podporuje digitální PTZ je k dispozici digitální režim. Ten funguje na principu digitálního zoomu a posun kamery je simulován pomocí posunu rámečku výřezu. Hlavní výhodou digitálního PTZ je možnost zobrazení a zazoomování na více míst v záběru kamery najednou (každý digitální PTZ ve svém okně, virtuální kamera).

Režim oblastí se používá v případě, když uživatel používá k pohybu kamery myš klikáním do obrazu kamery a kamera posune střed obrazu do bodu, kam klikl a případně zazoomuje. Dále je možné s kamerou pohybovat po krocích 5° osmi směry plus do nastavené domovské polohy (toto není výchozí poloha 0,0). Některé kamery také umí zařazovat PTZ požadavky do front. K dispozici je také OSD menu pro jednoduchou uživatelskou konfiguraci PTZ zařízení.

Příklad relativního PTZ:
V2 i v3: http://<servername>/axis-cgi/com/ptz.cgi?rpan=-2&rtilt=10
Příklad absolutního PTZ:
V2 i v3: http://<servername>/axis-cgi/com/ptz.cgi?pan=20&tilt=60
Příklad kontinuálního PTZ vlevo rychlosti 10 a dolů rychlostí 5:
V2 i v3: http://<servername>/axis-cgi/com/ptz.cgi?continuouspantiltmove=-10,5

2.2.8 Události (Events)

Některé kamery obsahují podporu událostí. Událost je v podstatě zpráva o něčem, co se stalo a mohlo by to být zajímavé pro uživatele nebo správce a měl by o ní být okamžitě informován. Událostí a možností reakce na ně je několik a liší se podle modelu kamery. Události mohou být spouštěné jak hardwarově tak softwarově.
Mezi události patří:

- Vstupní porty – změna stavu na vstupním portu
- PIR senzor – infračervený senzor detekoval pohyb
- Ruční spuštění – virtuální port pro uživatelské události
- Detekce pohybu – kamera detekovala pohyb v kontrolované zóně
- Detekce zvuku – kamera detekovala nastavenou hladinu hluku
- Manipulace s kamerou – pokud o krádež kamery
- Teplota – okolní teplota je mimo rozsah pracovních teplot
- Ztráta videa – pouze enkodér, ztráta analogového signálu
- Bootování – zařízení je připraveno například po vypnutí napájení
- PTZ pozice – kamera dosáhla požadované pozice
- Síťové připojení – kamera je opět připojena do sítě
- Detekce překročení čáry – speciální případ detekce pohybu

K těmto událostem lze registrovat reakce na ně:

- FTP upload – nahrání snímku z kamery na nastavený FTP server
- HTTP upload a oznámení – nahrání snímku na http server a odeslání zprávy na server
- E-mail – odeslání snímku z kamery se zprávou
- TCP/IP oznámení – odešle na TCP server zprávu o události
- Digitální výstup – nastavení dat na výstupní port
- Jdi na PTZ pozici – kamera se natočí do nastavené PTZ pozice
- Zahaj hlídku – kamera začne procházet přednastavenou hlídací trasu
- Automatické sledování – kamera zahájí sledování pohybu
- Rozsvícení – kamera zapne světlo, pouze VAPIX v3

Události jsou dynamické parametry, které lze přidávat pomocí příkazů na přidání parametrů. Při přidávání zvolíme, na jakou událost chceme reagovat a jak, například detekce pohybu a nastavení výstupního portu pro rozsvícení světla, nebo nabootování a odeslání TCP zprávy, že je kamera k dispozici.

Každá událost a reakce vyžaduje odlišné parametry, které jsou popsány v dokumentaci k parametrům kamery, protože jsou různé parametry v různých skupinách, je toto nastavování velmi nepřehledné. Parametry pro reakce jsou vždy ve skupině Events ale akce vyvolávající událost se nastavují v parametrech zařízení, které událost vyvolá.

2.2.9 Detekce a sledování pohybu

Detekovat pohyb lze v různých prostorech obrazu. Každý prostor lze definovat v parametrech a každý prostor může mít různou velikost a nastavení jako například úroveň detekce. Detekovat pohyb tak lze nastavit do prostoru dveří a reagovat na tento pohyb událostí. Bohužel i v případě PTZ kamer je okno pro detekci pouze v rámci obrazu a natočení kamery na něj nemá vliv, což je trošku škoda.

Jako reakce na detekování pohybu může být sledování objektu, který se pohybuje. Tato možnost je dostupná pouze u PTZ kamer. Sledování je možné zapnout pouze ve stanovených časech, např. po uzavření obchodu, kdy se po něm nepohybuje mnoho lidi a je vhodné je sledovat. Sledovaní je také možno vypnout v určitých oblastech kde se pohyb očekává. V případě sledování je už oblast vztažena k poloze kamery a natočení. Jedná se tedy o polohu skutečné oblasti v prostoru a ne polohu v obraze. Oblast se definuje pomocí PTZ souřadnic, ty lze vypočítat pomocí metody rozhraní *ptzcoodcalc*. Této metodě se předají souřadnice bodů v obraze a její odpovědí jsou PTZ souřadnice

těchto bodů. Lze také nastavit oblast, ve které se má objekt sledovat jako minimální a maximální hodnoty natočení a naklonění kamery.

Mezi speciální případ detekce pohybu je překročení čáry. V obrazu kamery je vytvořena virtuální čára. V případě překročení této čáry se vyvolá událost překročení čáry. Například kamera sleduje provoz na silnici, kde je zákaz předjíždění a v případě přejetí čáry spustí událost.

2.2.10 I/O

Speciální vlastností kamer jsou I/O porty, které obsahují pouze některé kamery. Obsahují vstupní a výstupní port, které jsou oddělené (např. 2 pro vstup a 2 pro výstup, celkem 4). Porty jsou pouze jednobitové a je tedy možná pouze sériová komunikace nebo pouze stav 0 nebo 1. Vstupní porty lze aktivovat, číst a monitorovat. Při čtení je vrácena aktuální hodnota na portu, při monitorování je výsledkem HTTP proud stavu požadovaných portů, je tedy možné získat časový průběh signálu. Stavy jsou / \ H a L. Výstupní porty mají stejné operace jako vstupní, navíc obsahují operace pro zápis na ně.

Speciálním případem portů jsou sériové porty. Ty jsou pro sériovou komunikaci vhodnější než obecné I/O porty, ale nehodí se pro spínání nebo generování signálů. Rozhraní k nim je velmi jednoduché. Nejprve je nutné port otevřít, otevřený zůstane po dobu, než ho uzavřeme. Pro čtení se předá počet bytů, které chceme přečíst a čas jak dlouho se bude na tyto data čekat (max 9s). Odpovědí jsou data z portu. Pro zápis je možné poslat data v hexadecimálním tvaru, nebo řetězec znaků o maximální délce 128.

Příklad dvou pulzů o délce 300ms a 500ms mezi nimi na portu 1:
V2: http://<servername>/axis-cgi/io/output.cgi?action=1:/300\500/300
V3: *http://<servername>/axis-cgi/io/port.cgi?action=1:/300\500/300*

2.3 Shrnutí

Zjistili jsme, že rozhraní pro IP kamery je několik a jsou poměrně obsáhlá. Rozhraní PSIA je jednodušší než OVIF ale je podporováno méně produkty. Rozhraní VAPIX obsahuje mnoho možností pro ovládání AXIS kamer. Mezi ně patří pochopitelně získání snímku, videa a zvuku, je li kamera vybavena mikrofonem. Kamery, které obsahují zařízení PTZ, které otáčí a naklání kameru, obsahují také rozhraní pro ovládání tohoto zařízení několika způsoby. Mezi velmi zajímává rozhraní patří především rozhraní událostí, které informuje uživatele nebo aplikaci o události, která může být kritická, například zakrytí výhledu kamery. Dalším rozhraním, které není běžné v případě IP kamer, jsou I/O porty, které umožňují spínat externí zařízení.

3 Návrh

Knihovna je od počátku navržena jako rozšiřitelná a je rozvržena do několika logických modulů. Každý je samostatně rozšiřitelný bez úpravy jiného. V rámci modulu můžou být nutné různé úpravy, nebo externí konfigurace z aplikace, která bude knihovnu využívat.
Moduly jsou:

- Ovládání IP kamery (PTZ[2],snímky, nastavení parametrů, …)
- AV Streamy a kodeky (MJPEG, RTSP, FFmpeg, …)
- Události kamer (pouze návrh, není implementováno, není součástí zadání)
- Utility (pomocné třídy např. Recorder, CameraPool, …)
- Vnitřní struktury

Všechny třídy v modulech, kromě modulu utilit, vychází ze základních tříd, které dědí (BaseCodec, BaseCamera, …). Tyto třídy jsou abstraktní a nelze je tedy přímo použít. Třídy poskytují základní funkčnost, která je předpokládána pro všechny jejich potomky. Všechny třídy knihovny jsou umístěny v namespace *libipcam*.

3.1 Ovládání IP kamery

Tento modul obstarává ovládání PTZ systému, získávání logů, nastavování parametrů, pořizování snímků a přístup k již nastaveným třídám z jiných bloků. Nejdůležitější je třída *CameraInit*, která slouží k automatické detekci rozhraní, které kamera, ke které se připojujeme, podporuje. Pokud je přidána další implementace rozhraní (ONVIF, PSIA, atd.), je nutné tuto třídu upravit, aby se pokusila kameru inicializovat s novým rozhraním.

Jakmile je kamera inicializována, je možné ji získat z CameraInit a začít používat. Inicializační třída vrací ukazatel na základní třídu BaseCamera, která obsahuje instanci té třídy, která obsahuje implementaci rozhraní, které kamera podporuje. O který typ se jedná, je také možné zjistit z CameraInit. BaseCamera je navržena tak aby reflektovala funkce rozhraní VAPIX2 a VAPIX3 (které je nadmnožinou VAPIX2).

Jak již bylo zmíněno, třídy rozhraní dědí BaseCamera a reimplementují její metody. Metody, které nejsou implementovány, generují výjimku NOT_IMPLEMENTED a nedojde tedy k pádu aplikace. Některé funkce jsou dost komplikované (parametry odvozeny z rozhraní) a třída tedy obsahuje wrappery. Ty zjednodušují použití těchto funkcí. Speciálně pro PTZ, které obsahuje kolem 40 parametrů, je navrženo několik wrapperů, které je doporučeno používat pro běžné operace místo metody *PTZControl*. Pro náročnější operace je ovšem využití této metody nezbytné, ale doporučují prostudovat dokumentaci k rozhraní kamery (např. VAPIX).

Pro komunikaci mezi knihovnou a kamerou jsou určeny komunikátory, které vychází ze třídy *BaseCommunicator*. Ta je v podstatě pouze interface pro zjednodušení psaní dalších komunikátorů. Pokud bychom chtěli komunikovat novým protokolem, implementujeme nový komunikátor. Bude zřejmě nutné upravit i samotnou implementaci rozhraní kamery, a to z toho důvodu že v různých typech komunikace se můžou parametry jmenovat jinak. Například VAPIX komunikuje pomocí protokolu HTTP ale ONVIF komunikuje pomocí protokolu SOAP. Je tedy pravděpodobné, že se parametry v obou protokolech budou v názvech lišit.

[2] Pan-Tilt-Zoom mechanismus pro natáčení kamery a zoomování

3.2 AV streamy a kodeky

Streamy a kodeky jsou rozvrženy do několika tříd. Některé jsou určeny pro zpracování dat, jiné pouze pro uchování dat. Třída BaseAVStream obsahuje základní iterface pro implementaci všech streamů. Jsou v ní umístěny buffery, jeden pro audio a druhý pro video, které jsou tvořeny frontou, a protože stream je nutné číst nepřetržitě, je vhodné tento buffer plnit z jiného vlákna. Tuto funkčnost samozřejmě poskytuje většina knihoven pro příjem streamů, ale pokud by tuto vlastnost neměla, mohl by v budoucnu vzniknout problém. Tímto způsobem je možné číst data v neblokujícím režimu.

Třídy AVData a AVRawData jsou určené pro uchování dat. AVData udržuje komprimovaná data přečtená ze streamu, nebo ze souboru. AVRawData udržuje dekomprimovaná data. Pro převod mezi AVData a AVRawData slouží kodeky.

Všechny kodeky vychází ze třídy BaseCodec. Tato třída abstrahuje obecné chování kodeků a je navržena velmi obecně a mělo by jít bez problému do ní implementovat jakýkoliv kodek. Každý kodek má 2 vstupy a 2 výstupy. Pro dekompresi je vstup AVData a výstup AVRawData, pro kompresi je vstup AVRawData a výstup AVData. Některé kodeky (např. MPEG) mají při dekompresi několik snímků zpoždění, proto je nutné zkontrolovat dostupnost dat pomocí metody *dataReady*.
Kodeky je také možné spárovat. To je vhodné především při rekompresi (např. MPEG4 → H.264). Pokud jsou kodeky spárovány, automaticky si vymění dekomprimovaná data. Není tedy nutné se zabývat předáváním dat a kontrolou dostupnosti dat. Musíme ovšem dbát rad v dokumentaci. Přesto, že jsou spojeny dva kodeky v logicky jeden, stále jsou to dva objekty. Některé pomocné třídy proto vyžadují ke své činnosti kodek v režimu komprese (enkodéru) i když jsou spojeny. Například *Recorder* vyžaduje enkodér z důvodu přístupu k metodě *dataReady* pro kontrolu dostupnosti komprimovaných dat k zápisu.

Do této části je vhodné také zahrnout třídy pro práci s multimediálními soubory. Tyto třídy vycházejí ze třídy BaseFormat. Ta poskytuje rozhraní pro čtení a zápis souborů. Třídy můžou implementovat různé formáty jako např. MKV nebo AVI.

3.3 Události

Tato část je pouze návrh, protože není součástí zadání, není implementována ani základní funkčnost. Protože moderní rozhraní IP kamer umožňuje klientům (aplikacím) přijímat informace o událostech jako například detekce pohybu, jsou v knihovně navrženy třídy pro zachycení a zpracování těchto událostí.

Předpokladem je, že každý interface kamery obsahuje podobné události, ale systém, kterým se přijímají, může být odlišný. Proto je zde třída *BaseEventHandler*, která má za úkol zaregistrovat nebo jinak oznámit kameře (záleží na interface kamery) zájem o událost a její příjem. Rozhraní VAPIX podporuje odesílání zpráv pro události různými způsoby a pro tento případ je asi nejvhodnější využít odeslání na TCP server.

Ke zpracování událostí aplikací slouží třída *BaseEventCallback*. Pokud bychom chtěli přijímat a zpracovávat jakoukoliv událost, musíme zdědit tuto třídu a implementovat minimálně 2 metody *registerEvents* a callback události na niž chceme reagovat. Metoda registerEvents vrací bitové pole událostí které chceme přijímat. Tato metoda je volána EventHandlerem, po zavolání jeho metody *register*.

Jakmile jsou události zaregistrovány, bude po vzniku události na kameře a jejím odesláním do aplikace zavolána pro každou událost příslušná metoda ve třídě EventCallback.

3.4 Utility

Do této kategorie patří všechny třídy, které nejsou nezbytné pro činnost, ale dříve či později by byla jejich implementace nutná v aplikaci, která by knihovnu využívala. Tyto třídy tedy značně zjednodušují psaní programů využívající knihovnu. Sem spadají třídy *Recorder, GroupCamera* a *CameraPool*.

Třída *Recorder* se využívá při nahrávání streamů a umožňuje i dekompresi, při té je nutné zvolit správné nastavení kodeků, aby byla rekomprese možná v reálném čase, jinak by se plnil buffer až by přetekl a začaly se zahazovat snímky.

Třída *GroupCamera* slouží ke skupinovému ovládání kamer a především k PTZ příkazům. Lze ji tedy využít pro aplikace, které používají více kamer k prostorovému vidění. Zde je opět nutné vzít v potaz vůli v servomechanismu kamer a zpoždění na síti i v knihovně. To může v důsledku zapříčinit ztrátu synchronizace pozice kamer.

Třída *CameraPool* slouží k umístění kamer do poolu. Zde mohou být kamery pojmenovány a zařazeny do skupin. Tyto skupiny lze využít k získání skupinové kamery.

3.5 Výjimky

Protože je knihovna napsána v jazyce C++ je v ní v dost velké míře využíváno výjimek. Každá část má definovanou třídu, kterou používá k vytvoření výjimky. Všechny tyto třídy vychází z třídy exception z namespace std. Pro část IP kamer je vytvořena třída *CameraException*, pro kodeky *CodecException* a pro streamy *StreamException*. Každá tato výjimka používá výčtový typ k upřesnění výjimky a reimplementují metodu *what*, která vrací textový popis výjimky. Více tříd je vytvořeno především pro jednodušší detekci, ve kterém modulu k výjimce došlo.

3.6 Vnitřní struktury

Ty slouží k implementaci samotné knihovny a nejsou implicitně nabízeny pro použití v aplikacích využívající knihovnu. Tyto třídy jsou *CircularBuffer,Thread,Mutex*. Samozřejmě pokud píšeme rozšíření je velmi vhodné tyto třídy použít.

Třída CircularBuffer, je implementace kruhového bufferu, který se používá při příjmu streamů. Je možné do bufferu zapisovat jak po znacích tak sérii znaků, a pro buffer je také možné použít již alokovanou paměť.

Dále pak třída Thread, která je objektovým wrapperem pro POSIX vlákna (pthread). Její interface je obdobný jako v případě obvyklých objektových implementací vláken (např. QThread v Qt). Většina tříd, které využívají vlákna, jako například streamy, běží v detašovaných vláknech, a to především z důvodu jednoduššího uvolnění paměti po jejich ukončení (paměť se uvolní automaticky). Toto činí implementaci pomocí Windows vláken velmi obtížnou, protože tato vlákna nepodporují detašovaný režim.

Třída Mutex je objektovým wrapperem mutexu z POSIX vláken. Není kompletně implementována funkčnost, ale po většinu operací je dostačující. Je možné nastavit, že mutex má být rekurzivní. To znamená, že pokud se ho pokouší zamknout vlákno, které už tento mutex zamknulo, nevznikne deadlock a vlákno je vpuštěno do kritické sekce. Mutex je možné velmi jednoduše implementovat i pomocí mutexů z Windows.

4 Knihovny

V této části se zaměřím na popis použitých knihoven a jejich přínos a základní použití. Knihovny jsou použity především z důvodu zjednodušení implementace. Některé funkce poskytované knihovnami by značně překročily rozsah bakalářské práce. Proto jsou využity knihovny pro práci s videem a komunikaci s kamerami.

4.1 cURL

Program cURL slouží především pro komunikaci na webu. Podporuje mnoho protokolů a mezi nejpoužívanější patří HTTP, HTTPS nebo FTP. Podporuje také protokol RTSP a RTP, k tomu samozřejmě také patří RTCP, ale pouze v režimu interleaved RTP. To znamená, že RTP pakety jsou posílány přes RTSP server pomocí transportního protokolu TCP. Tento protokol není vhodný pro přenos videa a dokáže velmi zpomalit jeho příjem.

V mojí knihovně je použita knihovna *libcurl*. Tato knihovna je právě základ aplikace cURL. Používá se ke komunikaci s kamerou pomocí protokolu HTTP, je možné také použít HTTPS ale je nutné přeložit (nebo mít přeloženou) knihovnu libcurl s podporou SSL (OpenSSL). Kameře jsou zasílány pomocí HTTP protokolu příkazy, na které reaguje. Reakce bývá dost často prázdná HTTP odpověď, to neplatí pro případy čtení nastavení a parametrů.

V knihovně není použit žádný objektový wrapper jako například cURLpp[3]. Zejména proto, že v době implementace, nebyla žádná verze dostupná ke stažení.

Libcurl je využita při implementaci třídy HTTPCommunicator a MJPEGAVStream, ve kterých tvoří jádro jejich funkčnosti.

S touto knihovnou by neměl vzniknout problém na žádné platformě, protože jsou dostupné implementace pro všechny běžné OS (Linux, Windows, Mac OS X) i pro 64 bitové verze a je také dostupná i pro mnoho exotických operačních systémů. Překlad je navíc jednoduchý i pod Windows, kde bývá často problém s portovanými knihovnami z Unix-like prostředí. Je dostupná pod licencí MIT.

4.1.1 libcurl

S knihovnou se pracuje velmi jednoduše a poskytuje více rozhraní. To nejjednodušší se nazývá „easy“. Knihovnu je nejprve nutné inicializovat pomocí funkce *curl_easy_init*, tím získáme také handle komunikace. Pak můžeme nadefinovat různé parametry, které předáme do knihovny přes funkci *curl_easy_setopt*. Zde se definuje URL adresa, port, HTTP autentizace a mnoho dalších. Pokud nechceme výsledek komunikace zobrazit na standardní výstup, musíme také nadefinovat callback pro čtení dat. Výchozí callback posílá data právě na standardní výstup. Můžeme také doplnit další callbacky pro ostatní činnosti jako například příjem hlavičky HTTP. Při práci v C++ musíme ovšem dát pozor na to, že pokud je naše funkce, která slouží jako callback, ve třídě, musí být tato funkce statická. Po nakonfigurování můžeme HTTP požadavek odeslat zavoláním *curl_easy_perform*. Výsledek a statistiku komunikace získáme funkcí *curl_easy_getinfo*. Pokud je to třeba, můžeme si nechat zduplikovat cURL handle a použít ho v nové komunikaci. Toho moje

[3] cURLpp objektový wrapper libcurl pro C++

knihovna využívá při vytváření MJPEG streamu, kdy je veškeré nastavení uloženo právě v handle a duplikováním odpadá nutnost znovu cURL nastavovat.

4.2 FFmpeg

FFmpeg je kolekce několika knihoven (framework) a programů pro práci s multimediálním obsahem, soubory a zařízeními. Obsahuje mnoho kodeků, audio i video filtrů, podporuje také mnoho multimediálních formátů a internetových protokolů pro přenos multimediálního obsahu. Implementace je více vláknová a je thread-safe, podporuje také hardwarovou akceleraci dekomprese videa pomocí grafické karty pro vybrané kodeky a pro ni potřebuje externí knihovny (libva, libvdpau, DXVA2). Hardwarovou akceleraci je ovšem nutné povolit při překladu FFmpeg a poté i při inicializaci kodeku v aplikaci. K tomu ovšem neexistují žádné příklady. Tato akcelerace ale není pro moji knihovnu vhodná především proto, že akcelerované kodeky používají odlišný formát pixelů. Každý kodek má jiný a proto by bylo obtížné používat tento formát k analýze obrazu bez jeho konverze. Není vhodný ani pro rekompresi, protože enkodéry žádný z těchto formátů nepodporují a bylo by opět nutné provést konverzi. Touto konverzí bychom ovšem zřejmě ztratili výhodu rychlosti hardwarové akcelerace. Takže pro ni zbývá využití pouze při přehrávání, kdy konverzi do RGB provádí také grafická karta. Existuje i fork FFmpeg s názvem libAV, který ale už není plně kompatibilní.

Celý framework je rozdělen do několika knihoven, každá z nich má jednotné rozhraní:

- *libavcodec* – knihovna s kodeky
- *libavfilter* – knihovna s audio a video filtry
- *libavformát* – knihovna pro čtení a zápis multimediálních kontejnerů
- *libavutil* – pomocné utility
- *libavdevice* – knihovna pro práci s multimediálními zařízeními
- *libaswscale* – knihovna pro změnu velikosti a formátu snímků
- *libswresample* – pro konverzi zvukových vzorků

[1]

4.2.1 libavcodec

Tato část FFmpeg obsahuje funkce pro kompresi a dekompresi multimediálních dat. Tato data jsou audio, video a titulky. Pro každý typ dat existují specializované funkce, ale funkce pro práci s daty určitého typu jsou stejné. Například všechna video data se dekomprimují pomocí funkce *avcodec_decode_video* a audio pomocí *avcodec_decode_audio*. V průběhu času se knihovna vyvíjí a staré funkce jsou nahrazovány novými, a to velmi jednoduchým stylem, jednoduše za název funkce připíší číslovku. Funkce s nejvyšším číslem je tedy nejnovější. Tím je zajištěna zpětná kompatibilita, ale staré funkce jsou většinou změněny a tvoří wrapper nové funkce.

Všechna data kodeků jsou uloženy ve struktuře AVCodecContext, která se předává jako parametr do funkcí. Ještě než ale budeme moci začít používat kodek, je nutné kodek zaregistrovat. To je možné buď jmenovitě zvolené kodeky, nebo pomocí *av_register_all* zaregistrujeme všechny kodeky. Pak musíme vyhledat kodek, který chceme použít. Pokud budeme komprimovat, tak hledáme pomocí *avcodec_find_encoder* a pokud dekomprimovat *avcodec_find_decoder*. Poté je nutné ještě

kodek nastavit jako například vejikost GOP[4], bitrate, nebo počet B snímků. V této fázi také můžeme povolit hardwarovou dekompresi. Protože ale není nastavování nijak zdokumentováno a není ani v ukázkových kódech, musíme hledat ukázky u přehrávačů, které FFmpeg používají a využívají hardwarovou dekompresi, např. MPlayer. Pak už jen stačí kodek otevřít pomocí *avcodec_open*.

Teď už můžeme začít kodek používat. To už je velmi jednoduché, přečtená data vložíme do funkce pro kompresi/dekompresi. Je nutné také vložit ukazatel na datový typ integer, do této proměnné nám funkce vloží informaci o tom, jestli už je snímek dekomprimován a je dostupný. Některé kodeky mají několik snímků zpoždění, to znamená, že snímek je dostupný až po tom, co je do kodeku vložen další snímek nebo snímky. To platí především pro kodeky MPEG. Abychom tedy mohli získat poslední snímek nebo snímky, musíme vložit prázdný snímek a z kodeku dostaneme poslední snímek.

Po skončení práce s kodeky je vhodné uvolnit paměť alokovanou kodekem. Pro to stačí zavolat funkci *avcodec_close*. Žádné další funkce již není třeba, i když jsme alokovali kontext kodeku, avcodec_close uvolní i tuto paměť.

4.2.2 libavformat

V této části FFmpeg najdeme funkce pro práci s multimediálními kontejnery a síťovými protokoly. Opět se pro různé typy kontejnerů, a i pro protokoly, používají stejné funkce bez rozdílu. Můžeme jak číst data z kontejnerů, tak i do nich zapisovat. Funkce jsou opět postupně modernizovány a jsou přejmenovávány stejným způsobem jako v případě libavcodec.

Stejně jako v případě kodeků je také zde nutné zaregistrovat formáty, které chceme používat jmenovitě, nebo všechny pomocí *avformat_register_all*. Pokud budeme přijímat multimediální stream přes počítačovou síť, je nutné také inicializovat síťovou část knihovny pomocí *avformat_network_init*. Nyní máme všechny části připraveny a můžeme otevřít soubor nebo stream. To lze udělat více způsoby, ale nejjednodušší je použít funkci *avformat_open_input*. Ta také automaticky rozpozná formát souboru podle přípony, nebo můžeme formát specifikovat explicitně, pokud si funkce neporadí sama. Pokud otevíráme stream, tak funkci předáme adresu streamu i s URI scheme (např. rtsp://localhost/example). Port je nutné zadávat pouze při nestandardním čísle portu dané služby.

Jakmile máme otevřený soubor nebo stream je nutné v něm vyhledat, jaké datové streamy obsahuje a poznamenat si jejich identifikátory, protože podle nich pak budeme rozlišovat audio a video data. Data se čtou v tzv. paketech, které obsahují vždy tak velkou část dat, aby mohla být zpracována kodeky. Ke čtení slouží funkce *av_read_frame*. V případě síťových streamů, běží příjem dat v jiném vlákně a tato funkce čte data z bufferu. V případě některých síťových streamů, jako například RTSP, je možné tento stream pozastavit a znovu spustit. Ve výchozím nastavení se stream spustí ihned po jeho otevření.

V případě vytváření multimediálních souborů je postup trochu obtížnější. Soubor se otevře přes *avio_open2*. Dále je nutné vytvořit datový stream pomocí funkce *avformat_new_stream*, které je také nutné předat kontext kodeku. Jakmile jsou definovány streamy, zapíšeme do souboru halvičku přes funkci *avformat_write_header*. Nyní už můžeme zapisovat jednotlivé datové streamy funkcí *av_write_frame*. Pokud už jsme zapsali všechna data, zapíšeme patičku *av_write_trailer* a uzavřeme soubor *avio_close*.

[4] GOP – Group of Pictures, počet neklíčových snímků mezi klíčovými snímky

5 Implementace

V této kapitole se zaměřím na implementaci knihovny a problémy, které při ní vznikly. Knihovna je implementována v jazyce C++, a je umístěna v namespace libipcam. Pokud bychom chtěli rozšířit knihovnu, nové třídy musíme umístit právě do tohoto namespace. Při implementaci jsou použity také výše zmíněné knihovny.

5.1 Utility

Nejprve se podíváme jaké pomocné struktury a funkce jsou k dispozici a pomohou s impelemntací.

Protože při další implementaci budeme potřebovat vlákna, je v knihovně implementována třída Thread, která slouží pro jednodušší implementaci objektových vláken. Protože je implementace objektová a funkce, která tvoří tělo vlákna, je v našem případě v objektu, musí být tato funkce statická. To je způsobeno tím, že knihovna pthread nepodporuje objektový callback a proto by funkci nepředala ukazatel *this*. Protože ale pthread podporuje předání ukazatelů do těla vlákna, je vytvořena funkce *entryPoint*, která příjme ukazatel na objekt vláka *this* a v tomto objektu zavolá instanční metodu *run*. V této funkci je tělo vlákna.

Pro příjem streamu je nutné mít rychlý buffer, do kterého můžeme rychle zapisovat nová data na jeho konec a číst z jeho začátku. Proto je v knihovně třída CircularBuffer, která implementuje kruhový buffer. Buffer je prakticky tvořen frontou znaků a obsahuje 4 ukazatele, na začátek bloku paměti, na konec bloku paměti, na začátek bloku pro čtení dat a na začátek bloku pro zápis dat. Pokud se poslední dva zmíněné ukazatele překříží, může to znamenat buď přetečení, nebo podtečení bufferu. Aby bylo jasné, který případ nastal, ale taky z praktických důvodů, je k dispozici také počet znaků v bufferu. Do bufferu je možné zapisovat a číst z něho po znacích ale také po řetězcích znaků. Je také možné použít vlastní blok paměti, jinak si třída alokuje blok požadované velikosti sama.

Protože je knihovna více vláknová, je někdy nutné zajistit vláknu výhradní přístup k datům. K tomu je implementována třída Mutex. Ta je v podstatě pouze objektový wrapper mutexu z pthread. Mutex také podporuje rekurzivní zamykání, což způsobí, že vlákno, které mutex zamklo, ho může zamknout vícekrát a nevznikne deadlock.

5.2 Rozhraní IP kamer

Knihovna komunikuje s kamerami pomocí tříd rozhraní, které implementují různá rozhraní kamer popsaných v kapitole 2.1. Aby bylo možné vybrat vhodné rozhraní pro danou kameru je implementována třída *CameraInit*. Ta slouží právě k výběru vhodného rozhraní. Protože je ale v knihovně implementováno pouze rozhraní VAPIX verze 2, je spíše uvažována pro budoucí rozšíření. I přesto je vhodné tuto třídu využívat a to také z toho důvodu, že inicializuje parametry potřebné pro komunikaci s kamerou. To provádí tak, že se pokusí spojit s kamerou přes daný protokol (v případě VAPIX je to HTTP protokol) a snaží se získat parametry kamery (pro VAPIX metoda params.cgi). Pokud se to nepodaří, je generována výjimka s popisem, k jaké chybě došlo. Na základě výjimky pak může aplikace provést opětovný pokus (např. chybné jméno nebo heslo). Pro budoucí použití je také možnost definovat upřednostňované rozhraní pro komunikaci. Například kamery AXIS

s firmware 5.x podporují jak rozhraní VAPIX3, tak ONVIF. Proto je vhodné definovat toto preferované rozhraní.

Jakmile je zjištěno, že kamera podporuje dané rozhraní, je vytvořen objekt ze třídy, která ho implementuje. Taková třída vychází z třídy *BaseCamera*. Protože ale taková třída by obsahovala pouze interface (v C++ pure virtual) a aby nebylo nutné definovat a implementovat všechny metody, jsou už implementovány (třída tedy není pure virtual). Tyto metody ovšem slouží pouze k usnadnění implementace a po jejich zavolání generuje výjimku NOT_IMPLEMENTED. To umožní aplikaci detekovat takové neimplementované metody.

Takto získaný objekt slouží ke komunikaci s kamerou a zasílá jí příkazy. Protože je implementováno pouze rozhraní VAPIX verze 2 komunikuje přes protokol HTTP pomocí knihovny libcurl. Ta je zapouzdřena v třídě *HTTPCommunicator,* kterou je možno použít pro jednoduchou komunikaci po HTTP protokolu (je tedy vhodná i pro neimplementované VAPIX verze3). Do komunikátoru se vloží metoda, kterou chceme zavolat a dále také parametry, které jsou zakódovány do URL. Komunikátor odešle zprávu a získá výsledek. Pro většinu metod se jedná o prázdný text. Na třídě s rozhraním je potom kontrola komunikace a případné generování výjimky.

Nejsložitější částí rozhraní je ovládání PTZ systému. Ten obsahuje mnoho možností a parametrů, které lze nastavit. To zahrnuje i ovládání různých filtrů a clony. K nastavení parametrů a ovládání PTZ slouží třídy *PTZCommand* a *PTZControl*. Ty obsahují několik veřejně dostupných proměnných, které jsou po konstrukci nastaveny na předem definované hodnoty, které jsou mimo definiční obor jednotlivých parametrů. Pro typ integer se používá konstanta INT_MAX, pro typ float pak hodnota NaN. Tyto třídy se pak předají třídě rozhraní a ta podle takto nadefinovaných hodnot pozná, že je kameře nemá odeslat.

Objekty kamer můžeme umístit do poolu *CameraPool*, který kamery uloží pod jménem (např. „kamera_vstup“). Kamery je také možné seskupovat do pojmenovaných skupin. Takto seskupené kamery je možné získat do jediného objektu ze třídy *GroupCamera*. Jakékoli zavolání metody v tomto objektu způsobí distribuci příkazů mezi jednotlivé kamery skupiny. Tím lze například synchronizovaně otáčet více kamerami, nebo kamery restartovat. Protože jsou do skupinové kamery předány objekty, které implementují rozhraní, můžou být ve skupině kamery, které mají odlišná rozhraní. Zprávy jsou tedy zasílány přes objekty jednotlivých rozhraní a nikoliv přímo.

Pokud bychom chtěli doimplementovat další rozhraní, je nutné implementovat třídu, která vychází z *BaseCamera* nebo z už vytvořeného rozhraní (VAPIX3 může použít VAPIX2). Poté doplnit inicializační logiku do třídy *CameraInit* a doplnit identifikátor rozhraní do výčtu rozhraní. Tento identifikátor potom identifikuje typ rozhraní, které je instancionováno a je tedy možné provést cast (v C++ dynamic_cast) na typ rozhraní.

5.3 Data

V knihovně se zpracovávají převážně multimediální data, a to ve dvou tvarech – komprimovaném a dekomprimovaném. K uchování dat slouží třída AVData a AVRawData. Data uvnitř mohou být pouze ukazatele na data, která jsou mimo třídu, nebo si alokuje paměť a uloží si kopii dat. Pro komprimovaná data poskytuje FFmpeg funkci, která je duplikuje. Pro nekomprimovaná je nutné použít funkce pro kopii obrazových dat a ostatní parametry zkopírovat.

AVData obsahuje pouze jednoduché funkce pro vkládání a získávání dat. Naproti tomu AVRawData obsahuje také metodu pro konverzi formátu pixelů snímků například z YUV420 na

RGB24, a pro uložení snímku ve formátu PPM[5] (typ P6). Konverzi provádí knihovna libswscale, která je součástí FFmpeg. Tato knihovna umí také změnit rozlišení snímků, ale možnost změnit rozlišení není implementována. Obě tyto třídy jsou navrženy tak, že mohou nést jak video, tak audio data, ale je implementována pouze podpora pro video data. Typ dat je uložen ve třídě, a tak mohou jiné třídy, které tato data zpracovávají, zjistit, o jaký typ dat se jedná a podle toho se zachovat (například kodeky). Při změně formátu nebo rozlišení snímků, je nutné alokovat paměť pro nový snímek. Při tom je nutné dávat pozor na to, že kodeky čtou data po slovech (na i386 4bajty, na x64 8bajtů). Pokud není paměť zarovnána na tuto hodnotu, dochází ke čtení a zápisu mimo alokovanou oblast. Tuto vlastnost je možné vypnout při překladu FFmpeg.

5.4 Streamy

Knihovna podporuje příjem dvou druhů streamů. První je M-JPEG, který funguje jako HTTP PUSH, a druhý RTSP stream. Ten je implementován pomocí FFmpeg a M-JPEG je implementován pomocí knihovny libcurl, ale teoreticky je možné ho implementovat také pomocí FFmpeg. Knihovnu libcurl jsem zvolil z toho důvodu, že v době implementace tohoto streamu jsem neznal možnosti knihoven FFmpeg.

Při vytvoření objektu M-JPEG streamu se zduplikuje nastavení HTTP komunikace, které je nastaveno v objektu rozhraní. Tím se vyhneme nutnosti znovu tuto komunikaci nastavovat a nemusíme si pamatovat jednotlivé parametry. Kamera začne stream posílat po otevření příslušné adresy a komunikace vypadá jako běžná HTTP komunikace s tím rozdílem, že nemusí nikdy skončit tok dat z kamery. To záleží na parametrech streamu, kde lze definovat například počet snímků nebo počet sekund trvání. Pro čtení hlavičky a těla jsou nadefinovány pro libcurl callbacky, kde z hlavičky je nutné přečíst především Content-Type, který obsahuje boundary. Značka boundary ve streamu odděluje jednotlivé snímky a každý snímek obsahuje svou vlastní hlavičku, která obsahuje Content-Type a délku dat, ta je nutná pro čtení celých snímků z bufferu. Callback těla pouze zapisuje data do kruhového bufferu a poté se snaží v bufferu najít hranice snímků a takto nalezený celý snímek se vloží do bufferu snímků. Protože celý příjem běží v jiném vlákně je přístup k bufferu chráněn mutexem.

Stream RTSP je implementován pomocí knihovny libavformat z FFmpeg. Z tohoto důvodu třída RTSP streamu obsahuje více metod, aby bylo možné předat parametry streamu do kodeku. Otevření streamu se provádí zadáním jeho adresy ve formátu „rtsp://jmeno:heslo@server/stream“. Protože přístup ke streamům na kameře může být omezen a dostupný je tedy pouze po přihlášení uživatele, je v takových případech nutné do adresy tyto údaje zadat. V případě RTSP je musíme vyplnit z hodnot uložených v objektu kamery a nelze, jako v případě M-JPEG streamu, získat duplikací nastavení, protože systémy nejsou kompatibilní. Ihned po otevření začne kamera vysílat, to je implicitní nastavení v libavformat. Protože se jedná o živý přenos, neobsahuje hlavičky, které by udávaly, jaké datové proudy a kodeky obsahuje. Proto je nutné pomocí funkce analyzovat přijaté snímky.

V této fázi vzniká několik problémů. Pokud bychom nastavili, že stream nemá začít vysílat ihned, funkce pro analýzu dat tato data nezíská, nikdy nevrátí výsledek a zasekne se (čeká na data, která nikdy nepřijdou). Pokud ale necháme stream vysílat od začátku a nezačneme ho číst ihned, dojde k přetečení bufferu. A pokud ho pozastavíme, v bufferu zůstanou staré snímky. Tento problém je v knihovně řešen tak, že se stream otevírá nadvakrát. Při prvním otevření se pouze zanalyzují data,

[5] PPM – Portable Pixel Map

aby byl znám kodek a identifikátory jednotlivých datových proudů (audio a video) a stream se uzavře. Při druhém se otevře s explicitním parametrem, který způsobí, že stream nezačne okamžitě posílat data. Nejjednodušší možnost, vyprázdnění bufferu, sice existuje ale funkce je dostupná pouze vnitřně v FFmpeg, například po posunu času. Ale tato funkce nefunguje při živých vysíláních.

Po přijetí dat, která už jsou určená ke zpracování knihovnou, je nutné přepočítat pozice těchto dat v čase streamu. RTSP stream posílá čísla snímků vzhledem k jeho FPS, což je 90000. Video data mají ovšem jiné FPS, které je obsaženo v nastavení kodeku, a to se pohybuje v rozmezí 20 až 100 (nejčastěji 25,30,100). Pokud je FPS 100, kamera vysílá každý čtvrtý snímek a výsledkem je tedy 25 FPS. Pokud bychom pozici nepřepočetli a data ukládali bez rekomprese, výsledný soubor by obsahoval pouze několik snímků, ale délka záznamu by byla několik hodin. Proto je nutné přepočítat pozice v rámci streamu na pozice pro kodek.

Po tom, co aplikace chce přijímat stream, knihovna vytvoří nové vlákno a začne číst data. Po přečtení dat je nutné identifikovat, o který typ dat se jedná, jestli o audio nebo video. Audio data jsou zahazována, protože zpracování audia není implementováno. Z dat je vytvořen objekt AVData do kterého se uloží kopie a tento objekt se zapíše do bufferu, který je chráněn mutexem.

5.5 Kodeky

Kodeky jsou implementovány pomocí knihovny libavcodec z balíku FFmpeg. To umožňuje kompresi téměř jakýmkoliv běžně používaným kodekem. Protože se všechna data stejného typu zpracovávají stejnými funkcemi, je celá implementace ve třídě FFMPEGCodec. Ta rozšiřuje třídu BaseCodec, která obsahuje pouze obecnou logiku pro zpracování dat, a přidává několik metod, které jsou nutné pro správnou inicializaci knihovny s kodeky a kodeků samotných. Jsou to metody pro vyhledání komprimovacího kodeku a dekomprimovacího kodeku, nastavení koprese a otevření těchto kodeků.

Obecný kodek může být v režimu komprese nebo dekomprese a takový kodek je možné spárovat s jiným obecným kodekem, v opačném režimu. Po spárování si kodeky vzájemně nastaví ukazatele na své protějšky a mohou začít zpracovávat data. Spárovat lze libovolné implementace kodeků a není tedy nutné, aby se párovaly pouze kodeky ze stejné knihovny. Tato možnost je také podpořena tím, že třídy uchovávající data umožňují konverzi mezi datovými formáty jednotlivých knihoven.

Po vytvoření objektu kodeku je nutné nejprve kodek vyhledat a otevřít, poté je možno mu začít předávat data. Protože některé kodeky mají několik snímků zpoždění, je nutné být informován o tom, jestli už jsou zpracovaná data k dispozici. K tomu zde slouží proměnná, do které se zapisuje stav dekomprese. Podle této proměnné pak aplikace přes metodu dataReady zjišťuje dostupnost dat na výstupu. Toto platí jak pro kompresi tak dekompresi.

Pro dekompresi je možné použít libovolný kodek, protože FFmpeg sám dokáže nastavit dekomprimující kodek z dat uložených v souboru nebo i z analyzovaných snímků ze streamu. Není tedy problém dekomprimovat libovolný video obsah. Kamery dnes běžně posílají video data komprimovaná kodekem MPEG4 nebo novějším H264. Oba tyto kodeky byly bez problémů otestovány.

Kodeku pro kompresi je nutné nejprve nastavit vlastnosti komprese, vyzkoušené hodnoty jednotlivých parametrů jsou k dispozici v mé knihovně. Je nutné především nastavit rozlišení, jinak nebude možné kodek otevřít a skončí chybou. Jsou otestovány kodeky MPEG4 a RAW, které funguji bez problémů. Naproti tomu kodek H264 funguje velmi nespolehlivě s daty, která jsou přečtena ze

streamu, a takto komprimované snímky nelze zapsat do souboru, protože kodek nespočítá jejich pozici (všechny se zapisují na pozici 0).

S kodekem je také spojen výsledný formát souboru, jako je AVI, MKV nebo MP4. Tyto formáty je možné vytvářet třídou FFMPEGFormat. Ta je implementována pomocí knihovny libavformat. Je možné vytvořit prakticky jakýkoliv známý multimediální kontejner. Ten se definuje pouze správnou koncovkou v názvu souboru, do něhož chceme data zapisovat. Je také nutné formátu předat kontext kodeku, ze kterého se potřebná data zapíší do hlavičky kontejneru. Formát AVI byl otestován jako nejspolehlivější a také si poradí s výše zmíněnou chybou kodeku H264. Ostatní formáty buď způsobují pády aplikace, nebo uloží poškozený soubor, ale pro určité kombinace kodeků a formátů fungují bez problému.

Pro implementaci nového kodeku (nebo sady kodeků), jsou nutné především metody encode a decode. Ty obsahují kód pro kompresi a dekompresi snímků. Pokud může mít nový kodek zpoždění snímků, je nutné také reimplementovat metodu dataReady, která původně zpoždění neuvažuje a vždy vrací true (snímek je vždy dostupný).

5.6 Budoucí vývoj

Knihovna je navržena v daleko větším rozsahu než by bylo možné implementovat v rámci této práce. Obsahuje neimplementované nebo částečně implementované části, které nejsou nezbytně nutné pro funkčnost knihovny. Takovou části jsou například události, které by rozšířily možnosti využití knihovny, nebo doplnění streamů a kodeků o schopnost přijímat a zpracovávat audio data.

V knihovně také často dochází k alokaci a uvolňování paměti. Nejčastěji se to děje při příjmu streamů, kdy se pro přijatý snímek alokuje nová paměť a po jeho zpracování se opět uvolní. Toto by se dalo omezit při použití paměťových poolů. Při tom by bylo možné neustále používat už alokovaný kus paměti a nebylo nutné stále alokovat novou paměť a tím snižovat výkonnost knihovny.

Mezi implementovanými rozhraními je pouze částečně implementované rozhraní VAPIX2, a to v takové míře jak je vyžadováno zadáním této práce. Bylo by tedy vhodné doplnit implementaci tohoto rozhraní a zcela implementovat rozhraní VAPIX3, které je podporováno novými kamerami AXIS, které verzi 2 nepodporují. Aby mohla knihovna komunikovat i s jinými kamerami než AXIS, je vhodné také doplnit knihovnu o rozhraní ONVIF. K tomu by bylo vhodné použít externí knihovnu, která umožňuje komunikaci pomocí SOAP.

V rámci kodeků a formátů, je potřeba se zaměřit na podporu více kodeků, než jen RAW a MPEG4 a formátu AVI. Pokusit se také o zvětšení stability kodeků, která je velice závislá na použité verzi FFmpeg. Ta je vydávána pouze ve vývojových verzích, které jsou sice stabilní, ale čas od času některé obsahují drobné chyby.

6 Používání knihovny

Knihovna je použitelná na všech operačních systémech, pro které jsou dostupné pomocné knihovny. To umožňuje například i použití ve vestavěných zařízeních, ve kterých funguje i knihovna FFmpeg (např. Embedded Linux).

Všechny hlavičkové soubory jsou vloženy do hlavičky libipcam.h, která je tedy jediná, jakou musíme vložit. Všechny třídy jsou umístěny v namespace libipcam. To je především proto, aby byly všechny třídy logicky pohromadě, ale také proto, že se některé třídy jmenují shodně s třídami jiných knihoven (např. Thread).

Pro vytvoření instance třídy pro ovládání kamery doporučuji vždy použít třídu CameraInit, kterou umístíme do try-catch bloku kvůli zachycení výjimek při připojování (kamera neodpovídá, chybné přihlášení,…). Poté zjistíme, jaký typ rozhraní byl vytvořen a případně pomocí dynamic_cast přetypujeme. Tímto objektem již můžeme kameru ovládat a můžeme ho vložit i do poolu kamer.

Pro vytvoření streamu kamery je také vhodné využít metody třídy kamery a nevytvářet je ručně (v případě M-JPEG to ani není možné). Po vytvoření streamu doporučuji získat objekt kodeku, který je už inicializován a nastaven. Ruční inicializace není prakticky možná, protože nejsou poskytnuty k tomu potřebné informace. Po spuštění příjmu dat, můžeme tato data vložit do tohoto kodeku a nechat je dekódovat. Pokud bychom chtěli provést rekompresi, tedy změnit kodek nebo jeho nastavení, je vhodné si kodeky pro kompresi a dekompresi spárovat. Kódovaná data (komprimovaná i RAW) lze zapsat do souboru třídami formátů. V knihovně je v současnosti implementována pouze třída formátů FFMPEGFormát, které se předá pouze název souboru a formát se rozpozná automaticky podle jeho koncovky (.avi, .mp4, .mkv, …)

Někdy je také nutné stream nahrávat a k tomu slouží třída Recorder. Té se předává stream, kodek a formát, do kterého budeme data zapisovat. Kodek může, ale nemusí, být spárovaný. Kodek párujeme pouze v případě rekomprese (a to i v případě nahrávání do RAW) a do rekordéru vložíme z dvojice kodeků ten, který je v režimu komprese. Pokud chceme stream uložit tak jak jsme jej přijali, předáme pouze kodek pro dekompresi získaný ze streamu. Vložený kodek se použije pro zápis parametrů dat do hlavičky souboru.

6.1 Překlad

Pro překlad je dostupný soubor Makefile, který je možno použít jak ve Windows (MinGW), tak v Linuxu. Pro překlad je samozřejmě nutné mít již přeložené potřebné knihovny libcurl a FFmpeg. Knihovnu je možné přeložit jako statickou nebo dynamickou knihovnu. Ukázkové kódy jsou překládány tak, že vyžadují dynamicky přeloženou knihovnu.

Překlad knihoven FFmpeg je velmi jednoduchý jak na Windows, tak na Linuxu. Knihovny sice mohou využívat další knihovny (openjpeg, zlib, libx264, …), ale ty nejsou nezbytné pro běžné operace. Je ovšem vhodné mít k dispozici překladač YASM, protože některé části jsou psány v assembleru a bez tohoto překladače by bylo nutné udělat tzv. „zmrzačený“ překlad (crippled build). Při překladu je vhodné povolit detekci instrukčních sad procesorů a zarovnávání paměti. Další možností je zakázání (výchozím nastavením jsou povoleny všechny) některých kodeků, formátů a protokolů, které například nebudeme potřebovat. Jediná výhoda tohoto je, že překlad je rychlejší. Knihovnu je také možné přeložit staticky nebo dynamicky. Knihovny ale není nutné překládat vždy,

protože na Linuxu jsou dostupné v balících a na Windows jsou běžně dostupné ke stažení již přeložené. Důležité je použít verzi, která obsahuje funkci *avcodec_encode_video2*[6].

Pro překlad knihovny libcurl je zapotřebí knihovna zlib (kvůli kompresi HTTP). Konfigurace překladu trvá velmi dlouho a v některých částech se může zdát, že se zasekla. Kromě překladu pomocí Make je také možné použít CMake. Na Windows bývají problémy při překladu s OpenSSL a proto je vhodné překládat bez podpory SSL. Při překladu v MinGW je nutné použít místo programu make (např. z MSYS), mingw32-make (v tomto případě není konfigurace vyžadována). Minimální verze pro správnou funkčnost není známa, ale knihovnu jsem testoval s verzí 7.23.0.

6.2 Měření zpoždění

Pokud bychom chtěli přijaté snímky analyzovat, je určitě dobré vědět, jaká je prodleva mezi pořízením snímku kamerou a jeho zpracováním v knihovně. Proto jsem provedl několik testů, které mají za úkol tyto prodlevy změřit. Testy jsou 3 a pro 2 rozlišení snímků, celkem tedy 6 testů.

1. Test 1 – Měří zpoždění mezi pořízením snímku v kameře a příjmem snímku knihovnou.
2. Test 2 – Měří zpoždění mezi pořízením snímku v kameře a dekompresí snímku knihovnou.
3. Test 3 – Měří zpoždění mezi příjmem snímku knihovnou a jeho dekompresí.

Před měřením je nejprve nutné synchronizovat čas kamery s časem počítače. To jsem provedl pomocí protokolu NTP. Ten IP kamery AXIS podporují. Horší situace je ovšem na straně počítače, kde jsem použil operační systém Windows. Ten sice obsahuje synchronizaci s NTP serverem, ale provádí ji s menší přesností než je při měření potřeba [2]. Proto se měření zdařilo až na třetí pokus. Při tomto pokusu jsem použil externí program na synchronizaci času NetTime. V systému Linux by takový problém zřejmě nevznikl. Čas jsem synchronizoval s fakultním serverem guta. Také je nutné nastavit na kameře zobrazování času ve snímku s rozlišením 1/100s. Přesnost měření v testu 1 a 2 je tedy 1ms ale rozlišení je pouze 10ms.

Pro měření jsem vytvořil aplikaci, která je dostupná v ukázkových kódech. Ta vytvoří stream a po přijetí snímku si poznamená čas přijetí snímku v milisekundách a předá snímek do kodeku. Po dekompresi si opět poznamená čas v milisekundách. Jakmile stream skončí, po 300 snímcích, začnou se snímky ukládat do souborů ve formátu PPM. Do názvu souboru je vloženo rozlišení snímku, oba časy a pořadové číslo snímku.

Následně je nutné snímky analyzovat skriptem, který jsem vytvořil v jazyce Python. Ten z názvu souboru přečte informaci o rozlišení a čase. Protože je čas pořízení snímku zakódován v obraze snímku, je nutné ho pomocí metody OCR převést na text. K tomu je nutný program GOCR. Aby se ale nesnažil hledat text v celém snímku je snímek oříznut tak, aby obsahoval pouze text s časem. To se provede pomocí aplikace pamcut z balíku Netbm. Protože je ale text v obraze bílý na černém pozadí, je také nutné invertovat barvy programem pnminvert ze stejného balíku. To z toho důvodu, že GOCR rozpoznává černé znaky. Takto upravený snímek již můžeme převést na text. V GOCR je nutné potlačit vlastní databázi znaků a vytvořit novou. Jinak se špatně rozpoznají některé znaky. Z textu se vytvoří čas v milisekundách a následně se všechny 3 časy porovnají. Skript také vytvoří datové soubory pro program gnuplot, ve kterém byly vytvořeny níže uvedené grafy.

[6] Přidána 8. 2. 2012 – revize 52f82a11489af88960c8774c142cbde78063365f

6.2.1 Parametry měření

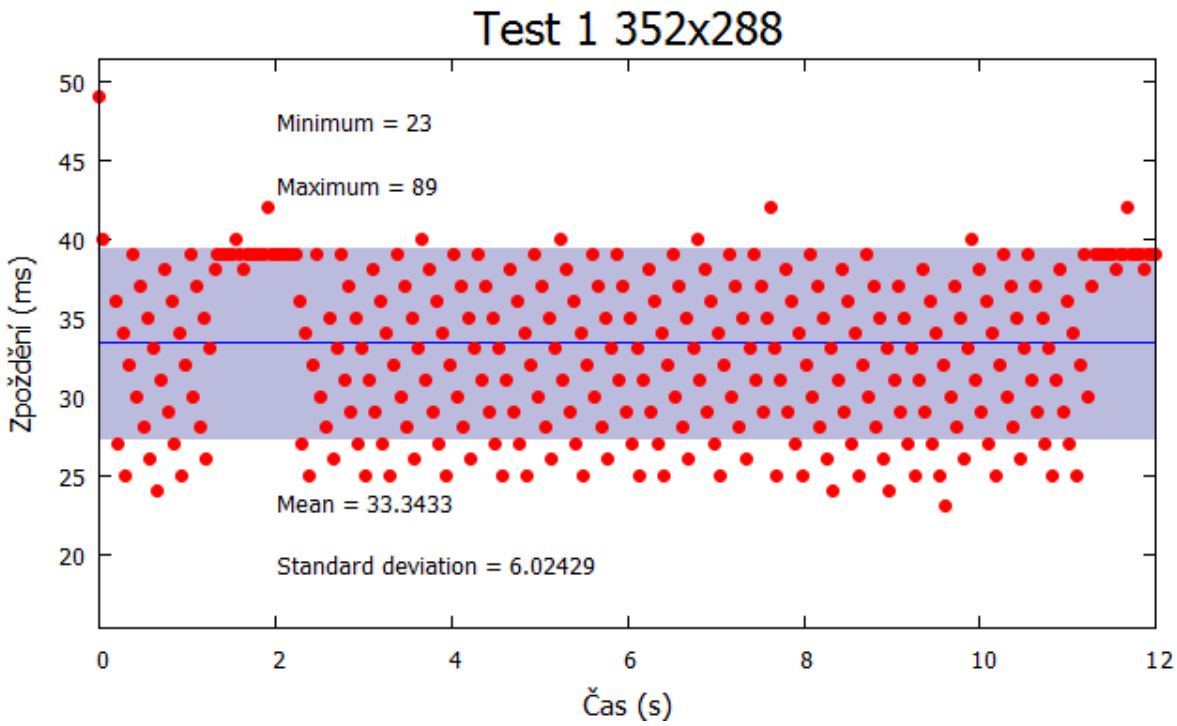
Počítač:

Procesor	Intel Core2Duo T8300 @ 2,4GHz
Paměť	4 GB
Operační systém	Windows 7 Professional x64 (aplikace 32bit)
Síť	Ethernet 100Mbit (kamera připojena přes jeden switch)

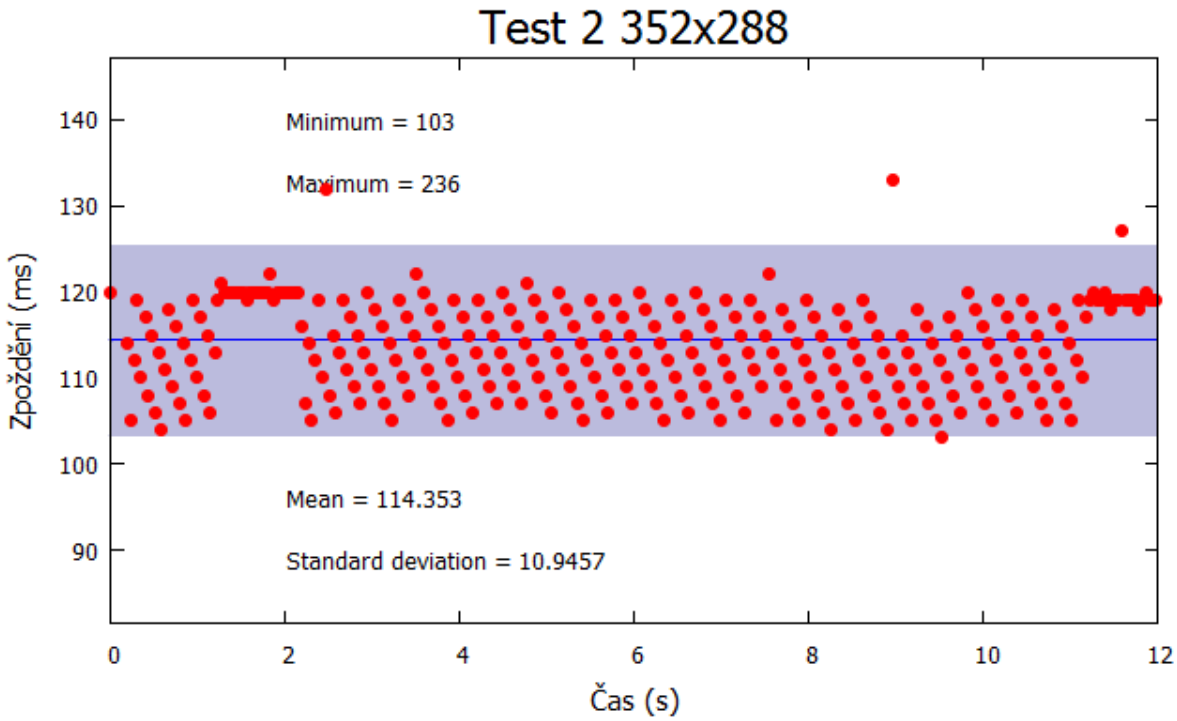
Kamera:

Výrobce	Axis Communications
Model	AXIS 214 PTZ

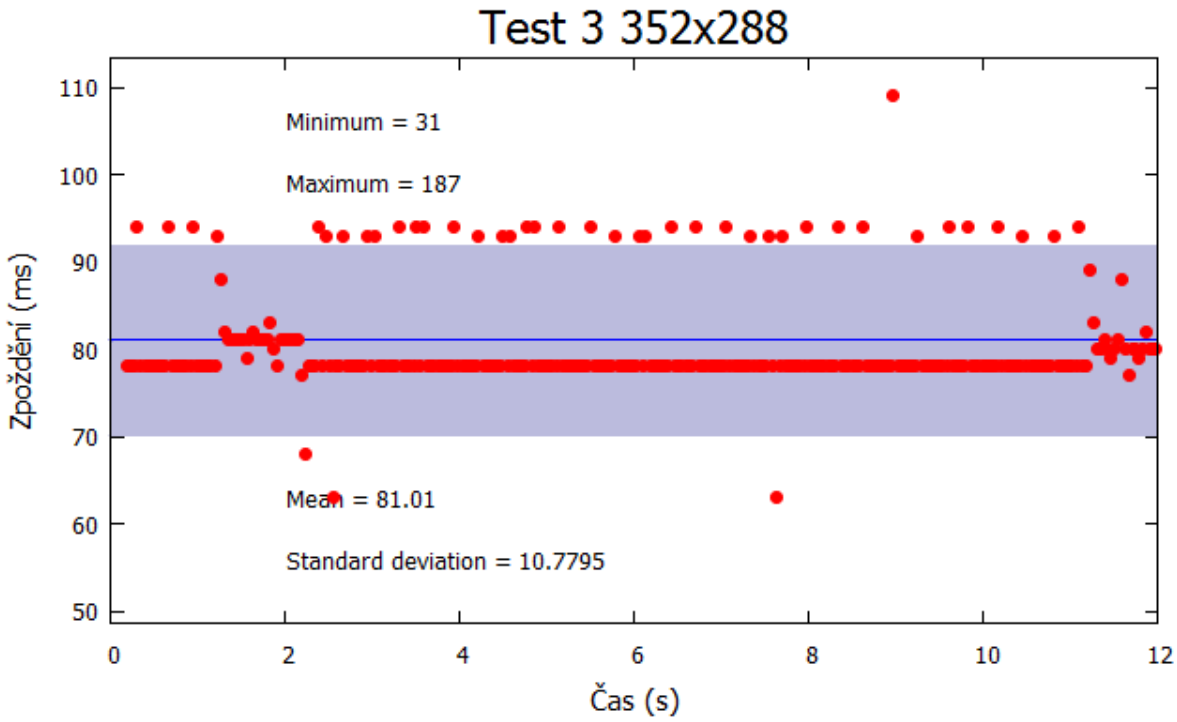
6.2.2 Výsledky měření



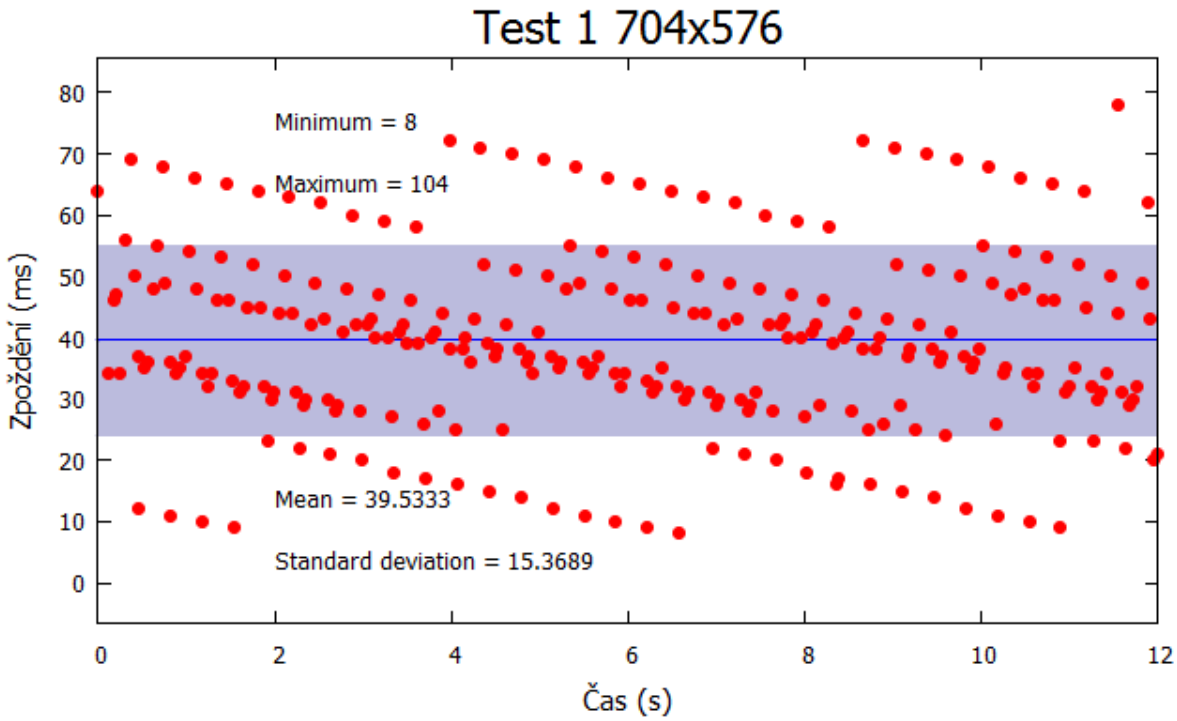
Obrázek 1: Zpoždění snímku mezi kamerou a knihovnou při rozlišení 352x288



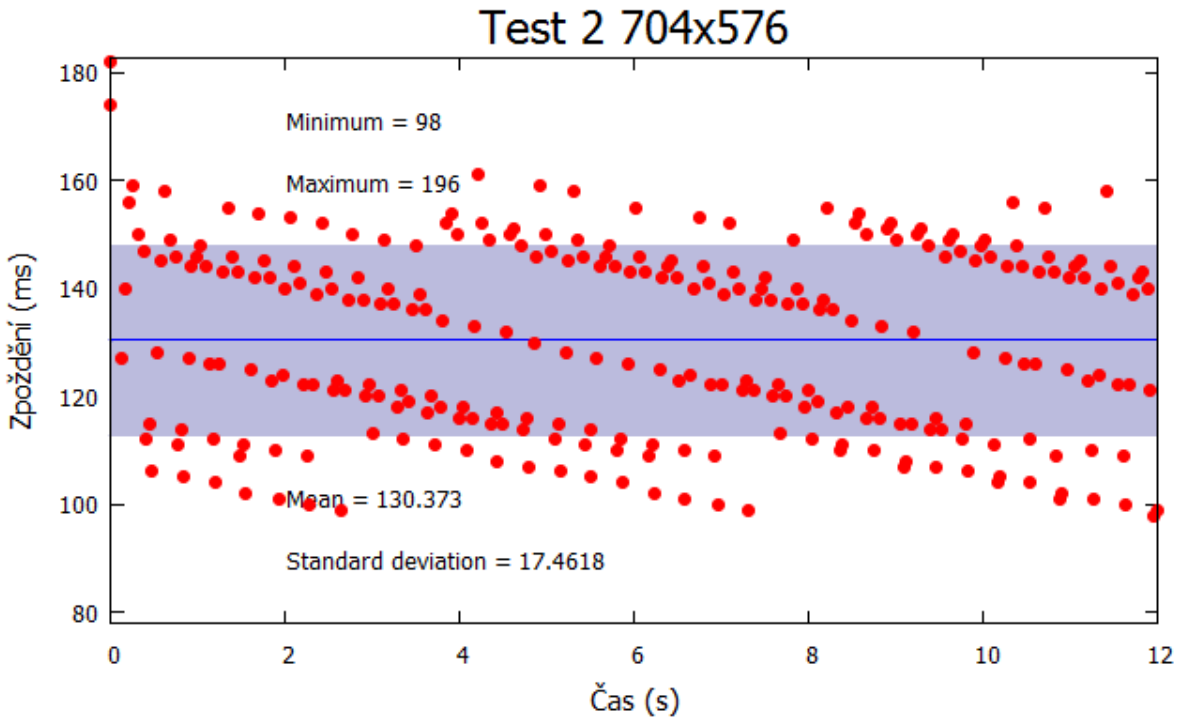
Obrázek 2: Zpoždění snímku mezi kamerou a dekompresí při rozlišení 352x288



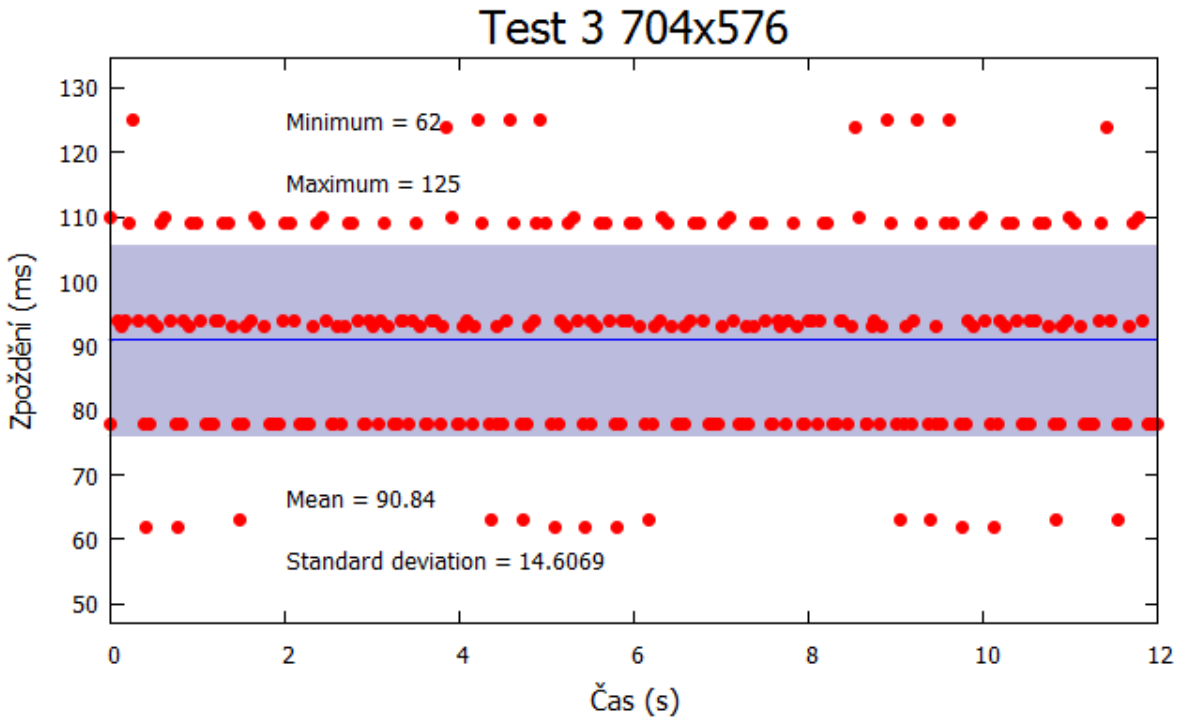
Obrázek 3: Zpoždění snímku při dekompresi při rozlišení 352x288



Obrázek 4: Zpoždění snímku mezi kamerou a knihovnou při rozlišení 704x576



Obrázek 5: Zpoždění snímku mezi kamerou a dekompresí při rozlišení 704x576



Obrázek 6: Zpoždění snímku při dekompresi při rozlišení 704x576

7 Závěr

Výsledkem mé bakalářské práce je nová a funkční knihovna, která může být použita v budoucích aplikacích. Poskytuje ty nejběžnější funkce pro práci s kamerami a vytváří možnost doplnění těch méně používaných. Umožňuje také doplnit další rozhraní IP kamer a tím rozšířit portfolio podporovaných zařízení. V knihovně je implementováno pouze rozhraní VAPIX2. To z toho důvodu, že jsem kamery s jinými rozhraními neměl k dispozici.

Pro vytvoření návrhu bylo nutné nastudovat dostupná rozhraní IP kamer. V návrhu jsem vycházel hlavně z rozhraní VAPIX s přihlédnutím k rozhraní ONVIF. To mi umožnilo co nejvíce abstrahovat rozhraní tříd tak, aby bylo použitelné pro jakékoliv rozhraní. Další důležité je rozhraní kodeků, které jsem vytvořil pro framework FFmpeg, který je použit pro multimediální část knihovny.

Tato část je, co se pochopení týče, zřejmě nejsložitější. Kodeky jsou velmi optimalizované, využívají specializované instrukce procesoru pro urychlení zpracování. Tyto optimalizace vyžadují pozornost, protože mají vliv i na vyšší úrovni programovacích jazyků. Je také nutné správě pracovat s pamětí, protože některé části kodeků využívají paměť stále dokola. Pro streamy je nutné dobře rozumět síťovým multimediálním protokolům, abychom chápali souvislosti mezi tímto protokolem a daty, která nese.

Knihovna umožňuje pohyb pozičním mechanizmem kamery (PTZ), částečnou konfiguraci parametrů a příjem multimediálních streamů z kamery. Mezi podporované streamy patří M-JPEG a RTSP. Video z nich je možné dekomprimovat a buď zobrazit uživateli, nebo analyzovat. Je také možné ho uložit do souboru v původním tvaru, tedy tak jak jsme jej přijali, nebo toto video komprimovat jiným kodekem. Teoreticky jsou podporovány všechny formáty souborů, ale nejspolehlivější je formát AVI. Z kodeků je také teoreticky možné použít libovolné kodeky, ale otestovány jsou MPEG4 a RAW. Teoretická možnost použít libovolný formát a kodek vychází z použití frameworku FFmpeg. Ale protože je ve hře mnoho parametrů, některé kodeky a formáty se nepodařilo zprovoznit.

Možností pro rozšíření je mnoho. Nejzajímavější je doplnění událostí a rozhraní VAPIX. Aby knihovna podporovala i jiné produkty než AXIS bylo by vhodné doplnit rozhraní ONVIF. Dále by bylo dobré rozšířit podporu kodeků a multimediálních kontejnerů. Pevně věřím, že tato knihovna poskytne dobrý základ pro budoucí aplikace.

Literatura

1. About FFmpeg. *FFmpeg.* [Online] FFmpeg, 7 April 2012. [Citace: 12. duben 2012.] Dostupné z: http://ffmpeg.org/about.html.

2. Hranice podpory konfigurace služby Systémový čas systému Windows pro prostředí s vysokou přesností. *Pomoc a podpora Microsoft.* [Online] Microsoft, 11. květen 2011. [Citace: 4. květen 2012.] Dostupné z: http://support.microsoft.com/kb/939322.

3. libcurl - programming tutorial. *cURL.* [Online] [Citace: 12. listopad 2011.] Dostupné z: http://curl.haxx.se/libcurl/c/libcurl-tutorial.html.

4. **Network Working Group.** Real Time Streaming Protocol (RTSP). *The Internet Engineering Task Force (IETF).* [Online] April 1998. Dostupné z: http://www.ietf.org/rfc/rfc2326.txt.

5. —. Hypertext Transfer Protocol -- HTTP/1.1. *The Internet Engineering Task Force (IETF).* [Online] June 1999. Dostupné z: http://www.ietf.org/rfc/rfc2616.txt.

6. —. RTP Profile for Audio and Video Conferences with Minimal Control. *The Internet Engineering Task Force (IETF).* [Online] July 2003. Dostupné z: http://www.ietf.org/rfc/rfc3551.txt.

7. —. RTP: A Transport Protocol for Real-Time Applications. *The Internet Engineering Task Force (IETF).* [Online] July 2003. Dostupné z: http://www.ietf.org/rfc/rfc3550.txt.

8. —. SDP: Session Description Protocol. *The Internet Engineering Task Force (IETF).* [Online] July 2006. Dostupné z: http://www.ietf.org/rfc/rfc4566.txt.

9. **AXIS Communications.** VAPIX® version 2,Event Handling API. *Axis Communications.* [Online] [Citace: 5. leden 2012.] Dostupné z: http://www.axis.com/files/manuals/eventHandling100326.pdf.

10. —. VAPIX® version 3, HTTP API. *Axis Communications.* [Online] [Citace: 5. leden 2012.] Dostupné z: http://www.axis.com/files/manuals/VAPIX_3_HTTP_API_3_00.pdf.

11. —. VAPIX®, Auto Tracking API. *Axis Communications.* [Online] [Citace: 5. leden 2012.] Dostupné z: http://www.axis.com/files/tech_notes/AutoTrackingAPI.pdf.

12. —. VAPIX®, HTTP API Specification. *Axis Communications.* [Online] AXIS, 17 October 2007. [Citace: 5. leden 2012.] Dostupné z: http://www.axis.com/techsup/cam_servers/dev/cam_http_api_2.php.

13. —. VAPIX®, RTSP API Specification. *Axis Communications.* [Online] AXIS, 29 October 2007. [Citace: 5. leden 2012.] Dostupné z: http://www.axis.com/techsup/cam_servers/dev/cam_rtsp_api.php.

14. **Nilsson, F.** *Intelligent Network Video: Understanding Modern Video Surveillance Systems.* 1st edition. Chelmsford : CRC Press, 2008. 416 s. ISBN 9781420061567.

15. General Documentation. *FFmpeg.* [Online] FFmpeg. [Citace: 4. únor 2012.] Dostupné z: http://ffmpeg.org/general.html.

Obsah CD

Na přiloženém CD se nachází následující soubory a složky:

- doc/ – složka s dokumentací (vygenerováno nástrojem Doxygen)
- examples/ – složka s ukázkovými kódy
- src/ – složka se zdrojovými kódy knihovny
- libs/ – složka se zdrojovými kódy potřebných knihoven
- text/ – složka s texty technické zprávy
- Makefile – soubor makefile pro přeložení knihovny (překládá jak staticky tak dynamicky)
- Doxygen – konfigurační soubor pro Doxygen
- LICENSE – text licence LGPL
- README – stručný postup pro překlad knihovny
- knihovna_pro_ovladani_IP_kamer.pdf – elektronická verze této technické zprávy